»Лекционен курс

»ООП1 (Java)

Наследяване (1) ›

Многократно използване

Въведение

- Многократно използване:
  - Много съществена идея, която предоставя повече от копиране и промяна на кода.
- В Java (както всичко друго) решението е свързано с класове:
  - Използваме повторно код за създаване на нови класове;
  - Без да ги създаваме от самото начало, а използваме съществуващи класове.

Въведение

- Принципно два подхода:
  - Композиция: В новия клас използваме обекти от вече съществуващи класове.
    + Използва се повторно функционалността на кода.
  - Наследяване: Новият клас се създава като тип на съществуващ клас:
    + Използва се формата на съществуващия клас;
    + Без да се променя съществуващия (базовия) клас;
    + Добавя се код .

Наследяване

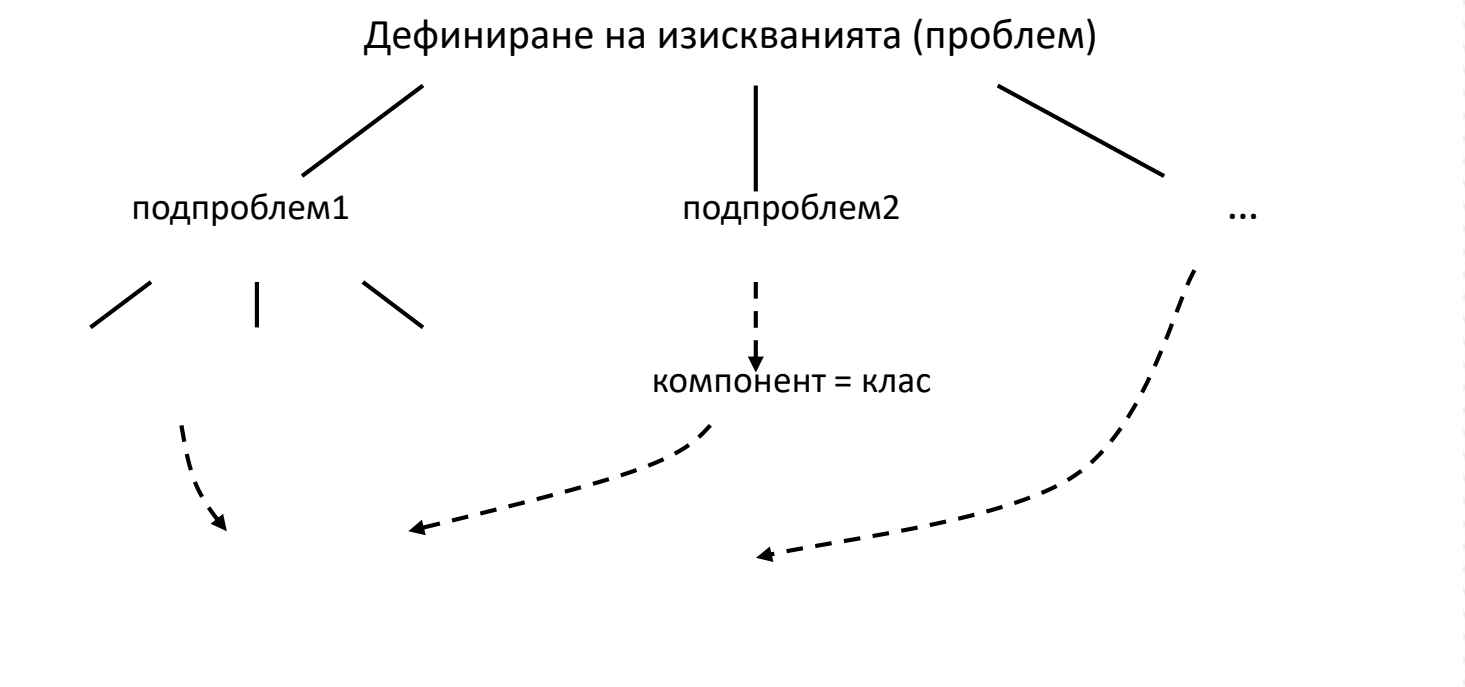
- Отношение между класовете.
- Елемент на моделиране в софтуерните архитектури.
- Класове: кога се разработват?
  - Най-късно: във фазата на проектиране чрез анализ на изискванията.
  - Цел: още при ‘анализ и дефиниция' класовете трябва да бъдат идентифицирани.

Софтуерен развой (фази):

- ***Анализ и дефиниция:*** *дефиниране на изискванията*
- ***Проектиране:*** *архитектура на софтуера*
- ***Реализация:*** *програма*

Как получаваме класовете?

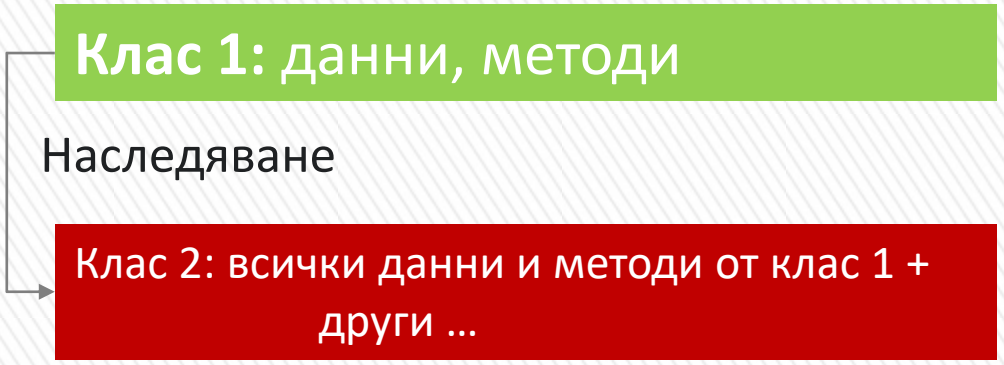
Чрез декомпозиране на проблема във фаза ‘анализ и дефиниция'



Софтуерна архитектура:
множество от компоненти (класове и др.) и техните отношения

Отношения между класове

Наследяване, асоциация, агрегация, …



По-общ

По-специален

Клас 1:

Наследява неговите свойства към клас 2.
По-общ от клас 2.
Суперклас на клас 2.

Клас 2:

Наследява …
По-специален …
Подклас …

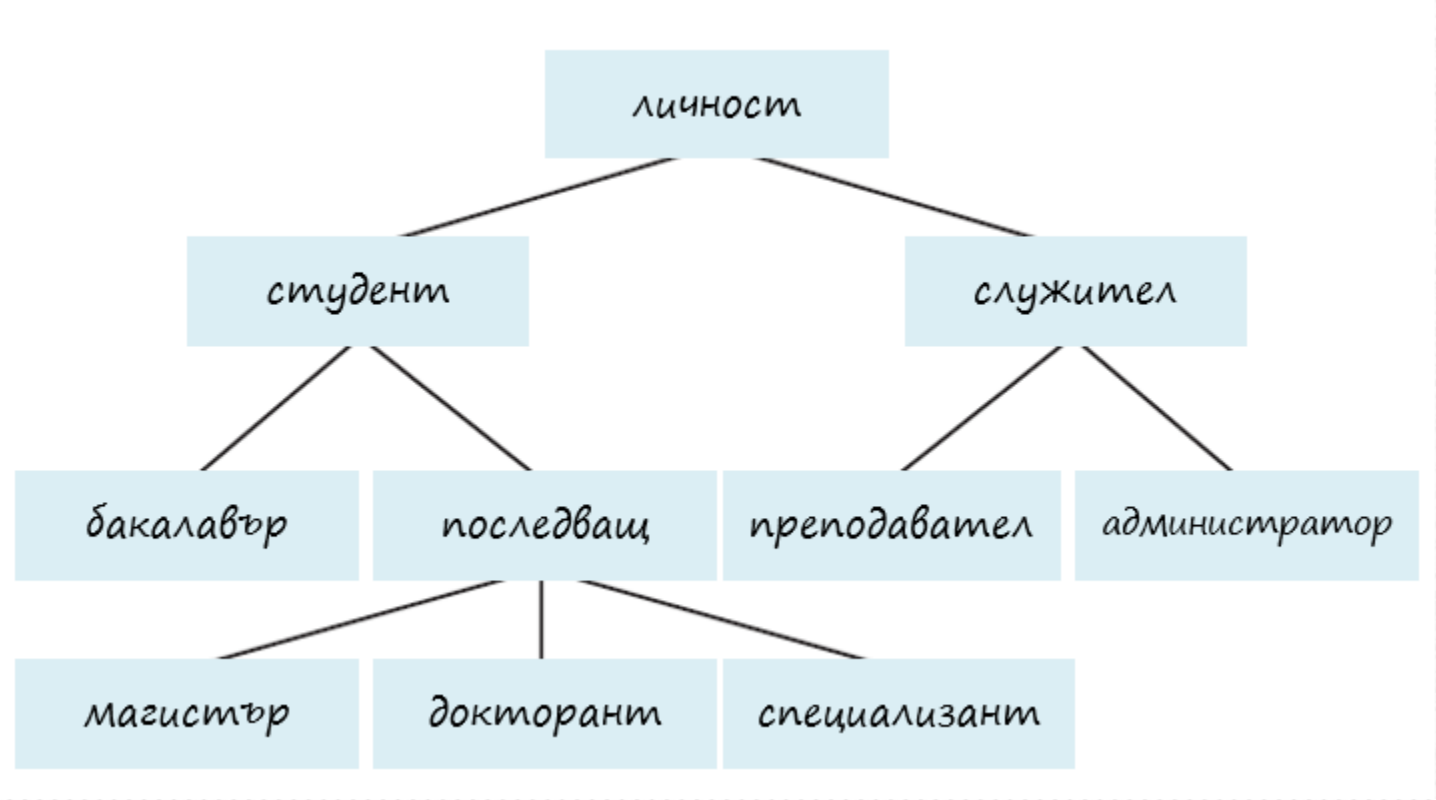
Синтаксис

SYNTAX

```
public class Derived_Class_Name extends Base_Class_Name
{
    Декларации на добавени променливи на обекти
    Декларации на добавени и променени методи
}
```

Въвеждащ пример

Въвеждащ пример



Въвеждащ пример

```
public class Person {
    private String name;
    public Person() {
        name = "Все още няма име";
    }
    public Person(String initialName) {
        name = initialName;
    }
    public void setName(String newName) {
        name = newName;
    }
    public String getName() {
        return name;
    }
    public void writeOutput() {
        System.out.println("Име: " + name);
    }
    public boolean hasSameName(Person otherPerson) {
        return this.name.equalsIgnoreCase(otherPerson.name);
    }
}
```

Какво прави програмата?

Въвеждащ пример

```
public class Student extends Person {
    private int studentNumber;
    public Student() {
        super();
        studentNumber = 0;
    }
    public Student(String initialName, int initialNumber) {
        super(initialName);
        studentNumber = initialNumber;
    }
    public void reset(String newName, int newStudentNumber)  {
        setName(newName);
        studentNumber = newStudentNumber;
    }
    public int getStudentNumber() {
        return studentNumber;
    }
    public void setStudentNumber(int newStudentNumber) {
        studentNumber = newStudentNumber;
    }
    public void writeOutput() {
        System.out.println("Име: " + getName());
        System.out.println("Факултетен номер: " + studentNumber);
    }
    public boolean equals(Student otherStudent) {
        return this.hasSameName(otherStudent) &&
               (this.studentNumber == otherStudent.studentNumber);
    }
}
```

Извиква конструктора на базовия клас

Какво прави програмата?

Въвеждащ пример

Коментар?

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Божур Цветков");
        s.setStudentNumber(123456789);
        s.writeOutput();
    }
}
```

Въвеждащ пример

Коментар?

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Божур Цветков");
        s.setStudentNumber(123456789);
        s.writeOutput();
    }
}
```

наследен метод

нов метод

препокрит метод

Въвеждащ пример

Какъв резултат?

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Божур Цветков");
        s.setStudentNumber(123456789);
        s.writeOutput();
    }
}
```

Въвеждащ пример

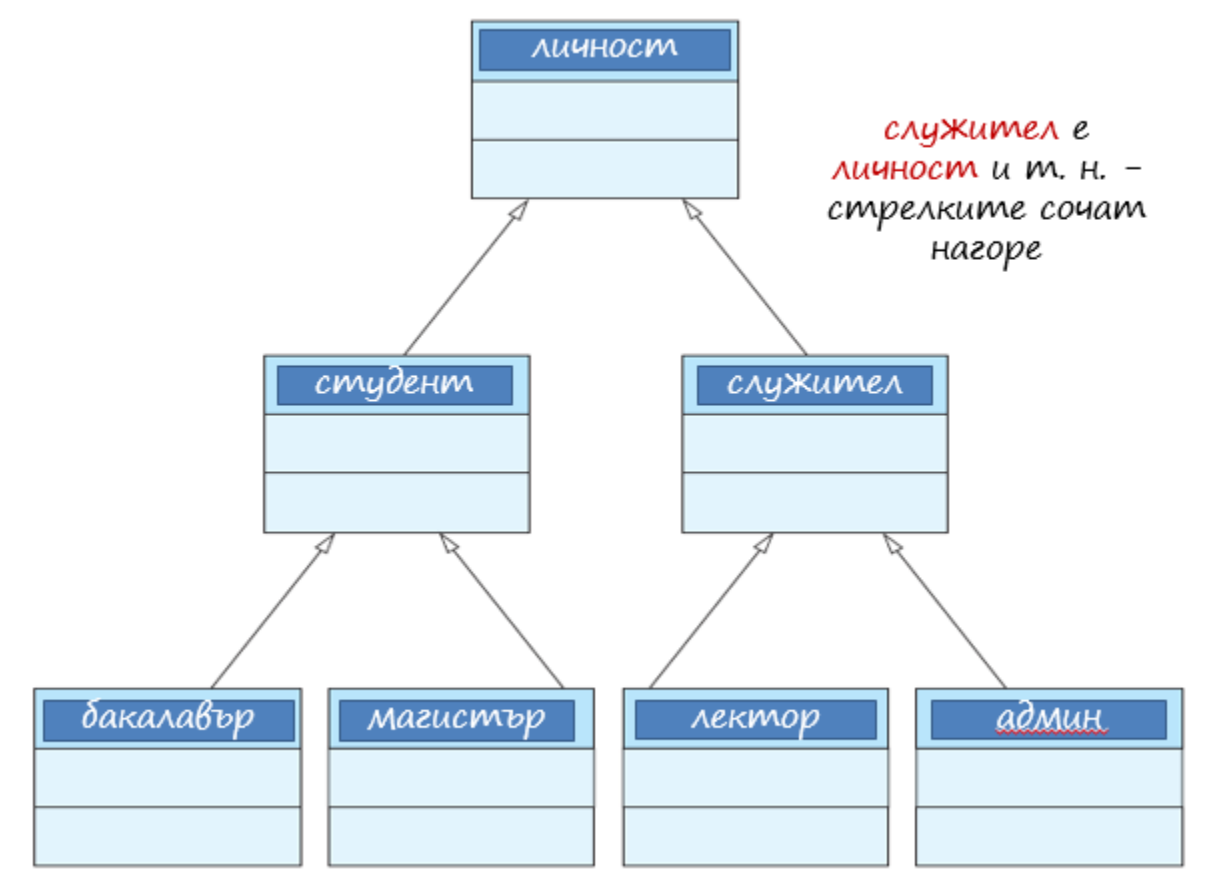
Какъв резултат?

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Божур Цветков");
        s.setStudentNumber(123456789);
        s.writeOutput();
    }
}
```

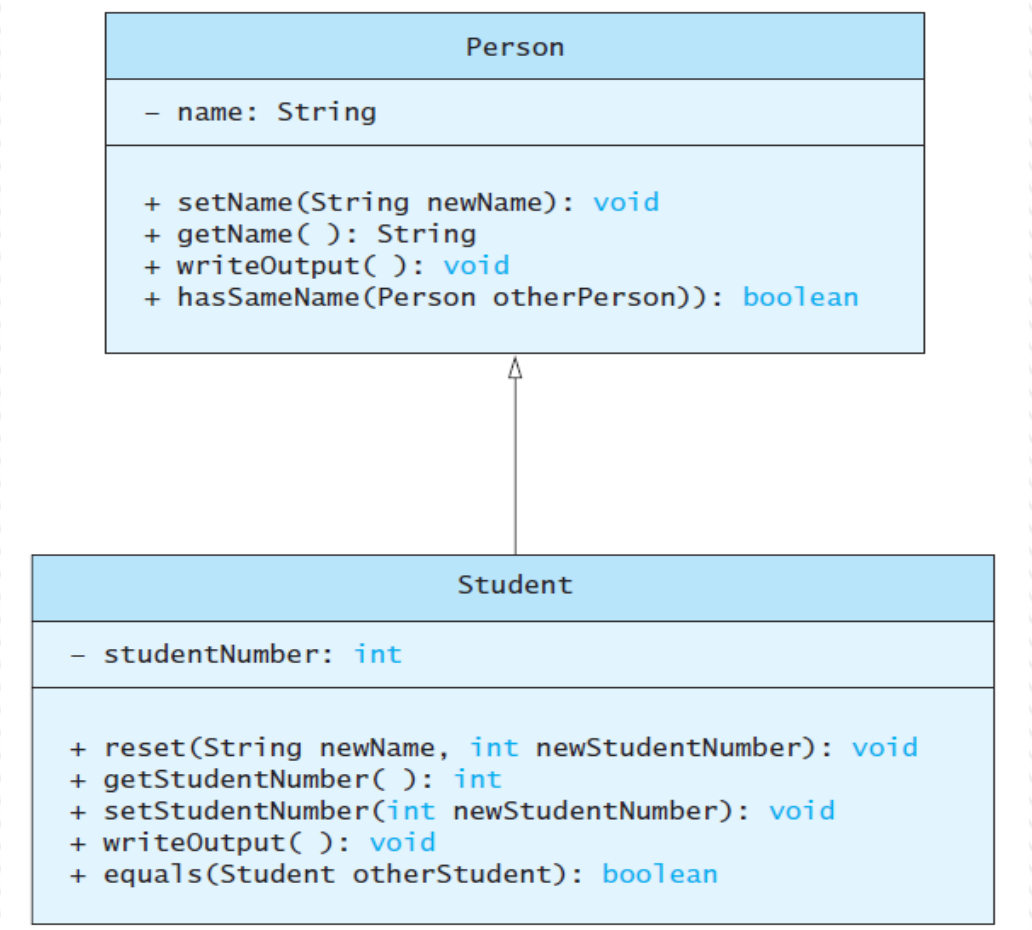
```
Име: Божур Цветков
Факултетен номер: 123456789

Process finished with exit code 0
```

Йерархия от класове

UML диаграма на йерархията от класове



Детайлна UML диаграма



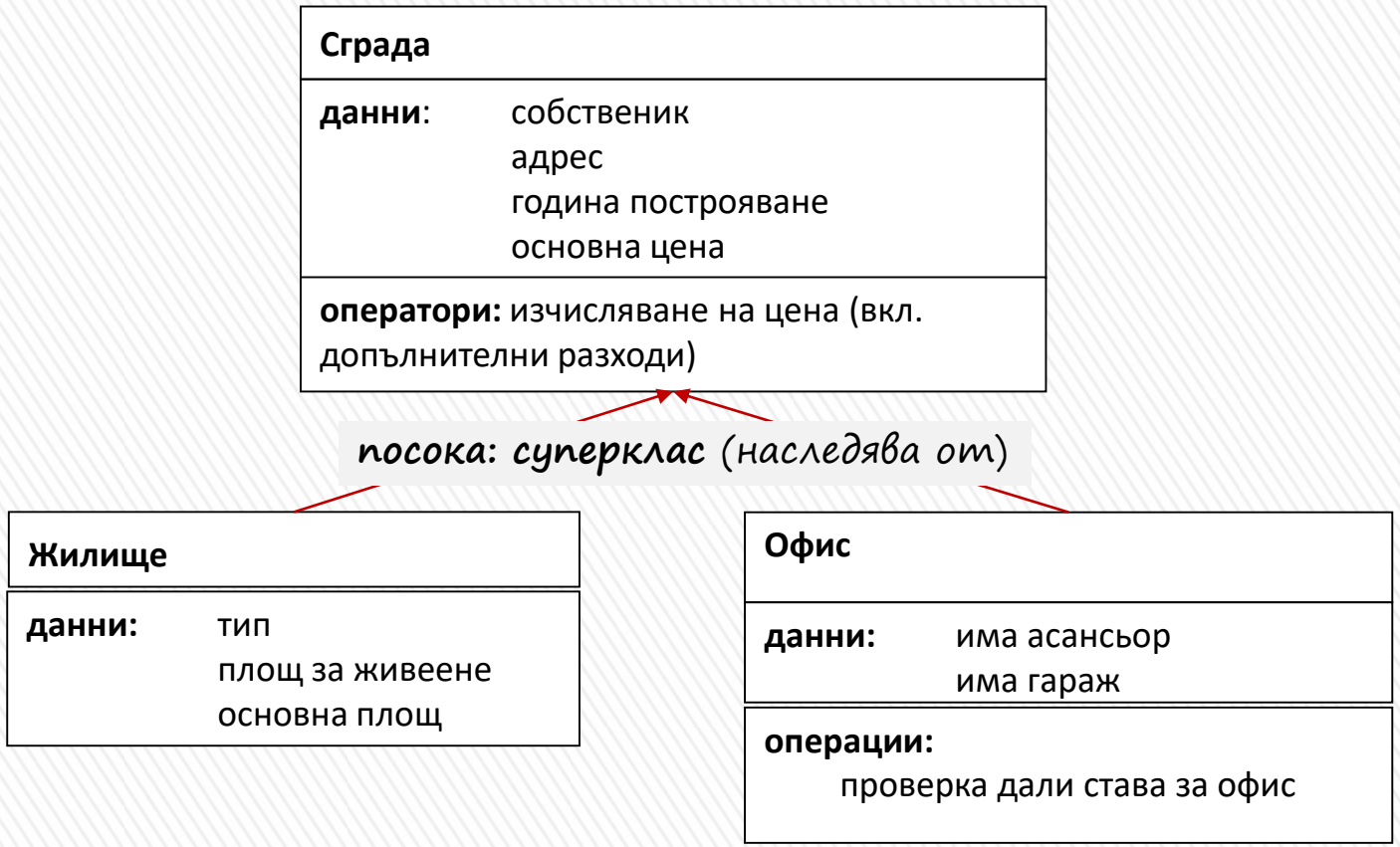
Добра практика

- Ако връзката is-a не съществува между два класа, не трябва да използваме наследяване за извличане единия клас от другия.
- Вместо това, трябва да помислим за дефиниране на обект от един клас в другия клас (тази връзка се нарича has-a).

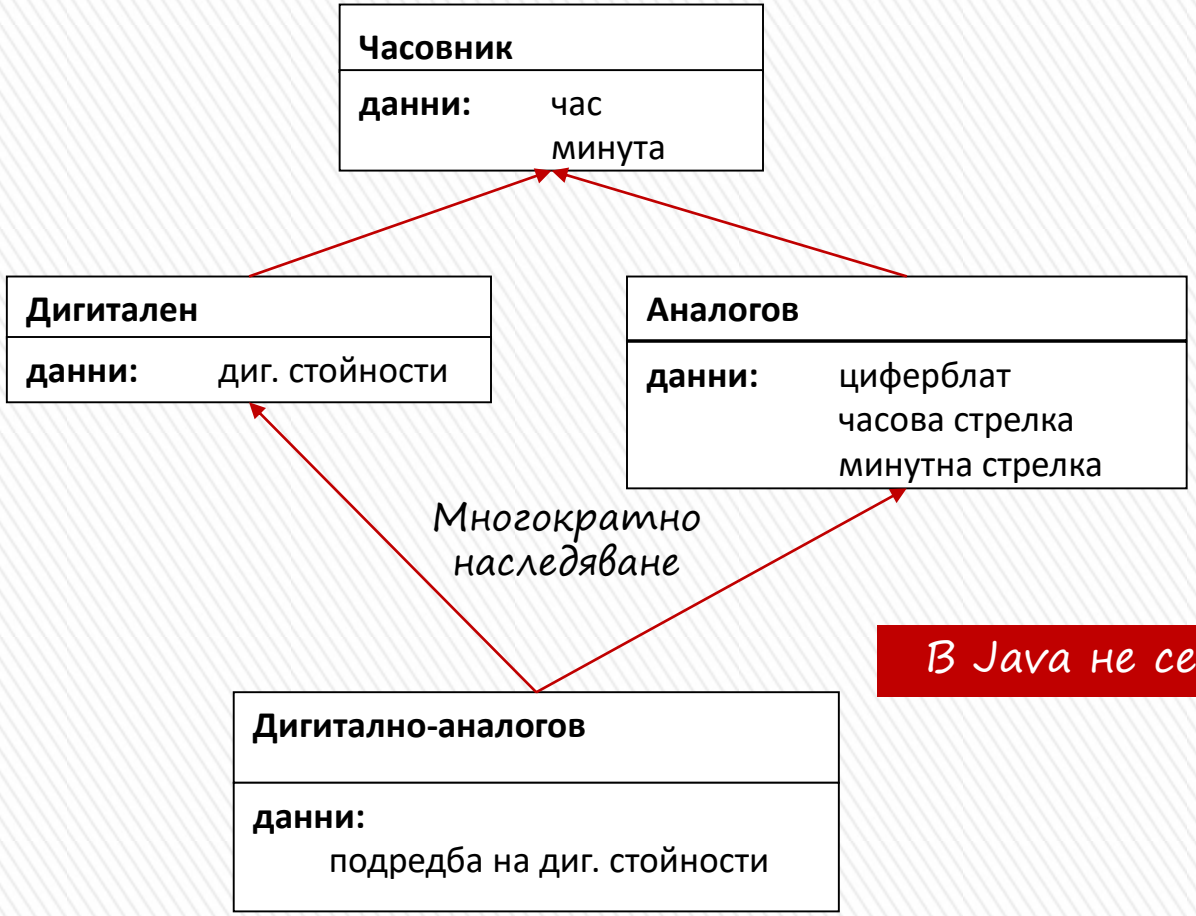
Пример за наследяване

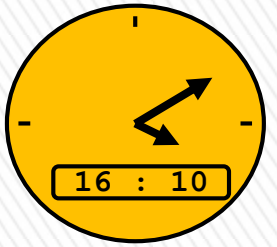
Управление на недвижимо имущество:

нотация: UML (език за графично описание на софтуерни архитектури)



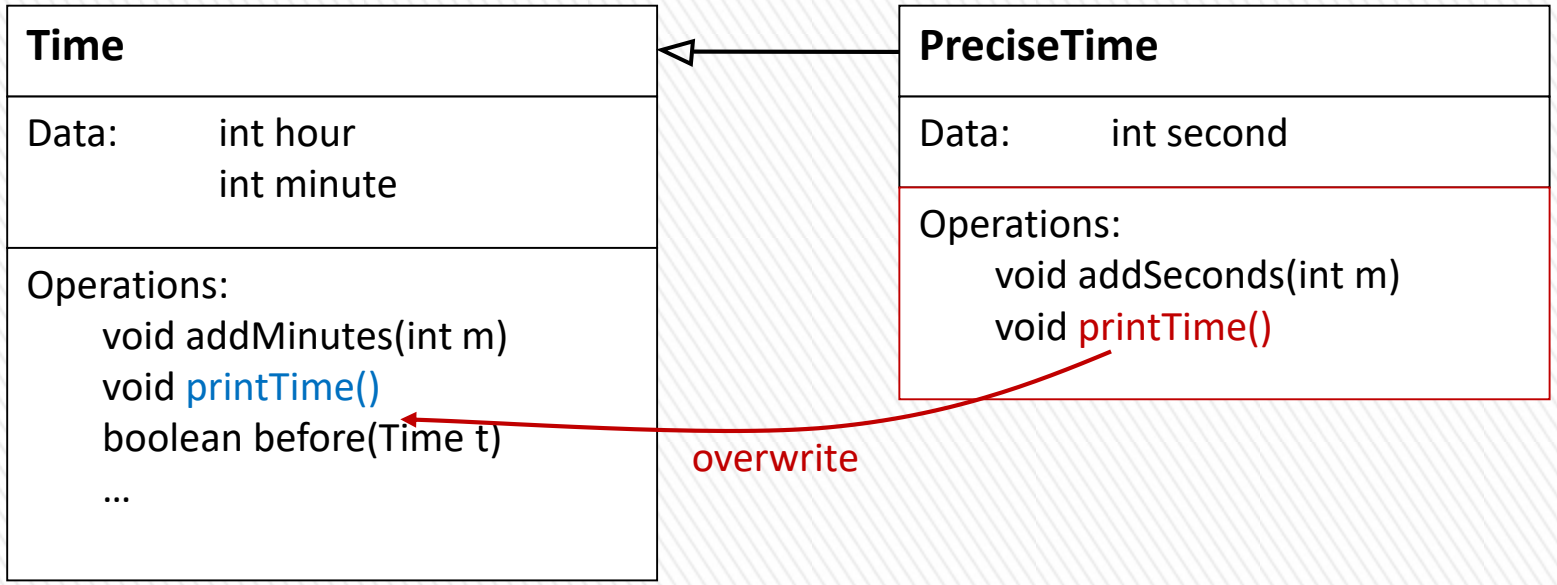
Пример за многократно наследяване





В Java не се поддържа (но в C++)

Часовник - прецизно време



Методи на подклас

Създаване методи за подклас

- При създаване на подкласове можем да специфицираме допълнителни данни и методи.
- При дефиниране на методи в един подклас съществуват следните три възможности:
  - > Препокриване на методи от суперкласа:
    - Специфицираме методи със същата сигнатура като в суперкласа.
  - > Наследяване методи от суперкласа:
    - Ако един метод не е препокрит, тогава автоматично се наследява.
  - > Дефиниране на нови методи:
    - Специфицираме нов метод, който не съществува в суперкласа;
    - Могат да бъдат използвани само за обектите на подкласа.

Препокриване на методи

Кой от двата?

```
public class Person {
  …
    public void writeOutput() {
        System.out.println("Име: " + name);
    }
…
}
```

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Божур Цветков");
        s.setStudentNumber(123456789);
        s.writeOutput();
    }
}
```

```
public class Student extends Person {
  …
    public void writeOutput() {
        System.out.println("Име: " + getName());
        System.out.println("Факултетен номер: " +
                           studentNumber);
    }
  …
}
```

- Извикването на s.writeOutput() ще използва дефиницията на writeOutput в класа Student, а не дефиницията в класа Person, тъй като s е обект на класа Student
- Когато препокриваме метод можем да променим тялото на дефиницията на метода, но не сигнатурата (заглавната част) му, включително типа на резултата

Препокриване на методи

Кой от двата?

```
public class Person {
   …
     public void writeOutput() {
          System.out.println("Име: " + name);
     }
…
}
```

```
public class Student extends Person {
  …
    public void writeOutput() {
        System.out.println("Име: " + getName());
        System.out.println("Факултетен номер: " +
                                studentNumber);
    }
  …
}
```

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Божур Цветков");
        s.setStudentNumber(123456789);
        s.writeOutput();
    }
}
```

```
Име: Божур Цветков
Факултетен номер: 123456789

Process finished with exit code 0
```

Препокриване на методи

В производен клас, ако включим дефиниция на метод, който има:

- Същото име;
- Точно същия брой и типове параметри;
- Същия тип резултат.

Като на метод, който вече е в базовия клас, тази нова дефиниция замества (препокрива) старата дефиниция на метода когато обектите от производния клас получат извикване на метода.

Т.е. двата метода (препокритият и препокриващият) трябва да имат една и съща сигнатура.

Препокриване и претоварване

- Не трябва да смесваме препокриване на метод с претоварването на метод.
- Когато препокриваме метод, новата дефиниция на метода в производния клас, има същото име, същия тип резултат и точно същия брой и типове параметри.
- Ако методите имат същото име, но различен брой параметри или параметър от различен тип ще бъдат претоварени.
- В такива случаи производният клас ще има и двата метода.

Препокриване и претоварване

Ако в класа Student добавим следния метод …

Коментар?

```
public String getName(String title) {
        return title + getName();
}
```

В класа Person съществува метод getName() …

```
public class Person {
  …
    public String getName() {
        return name;
    }
  …
}
```

Препокриване и претоварване

Коментар?

Ако в класа Student добавим следния метод …

```
public String getName(String title) {
        return title + getName();
}
```

Методите са претоварени, т.е. и двата са налични в Student

В класа Person съществува метод getName() …

```
public class Person {
  …
    public String getName() {
        return name;
    }
  …
}
```

Конструктори

- Наследените класове имат свои собствени конструктори
- Не се наследяват конструктори от базовия клас
- В дефиницията на конструктор на наследения клас първо действие трябва да бъде конструктор на базовия клас

Локални данни

Локални данни

Коментар?

Създаваме Student обект

```
Student joe = new Student();
joe.reset("Joesy", 9892);
```

Искаме да присвоим име на студент

```
public void reset(String newName, int newStudentNumber)
{
    name = newName;
    studentNumber = newStudentNumber;
}
```

name локална в Person

Локални данни

Коментар?

Създаваме Student обект

```
Student joe = new Student();
joe.reset("Joesy", 9892);
```

Искаме да присвоим име на студент

```
public void reset(String newName, int newStudentNumber)
{
    name = newName;
    studentNumber = newStudentNumber;
}
```

name локална

Невалидно!

```
public void reset(String newName, int newStudentNumber)
{
    setName(newName);
    studentNumber = newStudentNumber;
}
```

Валидно!

Локални данни

Базов клас Time

Коментар?

```
class Time {
    private int hour, minute;
    public Time() ...
    public void addMinutes(...)
    public void printTime()
}
```

Наследен клас PreciseTime

```
class PreciseTime extends Time {

    private int second;

    public PreciseTime(...) ...

    public void addSeconds(int s)

    public void printTime()
}
```

Нова променлива

Нов конструктор

Нов метод

Препокриване

Локални данни

Къде са видими hour, minute?

само в клас Time

```
class Time {
   private int hour, minute;
  ...
}
```

```
class PreciseTime extends Time {
   private int second;
   public void printTime() {
      if ((hour == 0) && (minute == 0))
         ...
      if (second < 10) ...
}
```

Видимост

Какво предизвиква използването на hour, minute?

```
class Time {
   private int hour, minute;
  ...
}
```

```
class PreciseTime extends Time {
   private int second;
   public void printTime() {
      if ((hour == 0) && (minute == 0))
         ...
      if (second < 10) ...
}
```

Грешка!

Видимост

Защо?

променливите са локални

```
class Time {
    private int hour, minute;
   ...
}
```

```
class PreciseTime extends Time {
    private int second;
    public void printTime() {
        if ((hour == 0) && (minute == 0))
            ...
        if (second < 10) ...
}
```

Грешка!

Видимост

Решение?

Видими във всички подкласове

```
class Time {
   protected int hour, minute;
  ...
}
```

```
class PreciseTime extends Time {
   private int second;
   public void printTime() {
      if ((hour == 0) && (minute == 0))
         ...
      if (second < 10) ...
}
```

Базови и наследени класове

Базов и наследен клас

За външния свят:

- Новият клас има същия интерфейс, като този на базовия.
- Евентуално някои допълнителни методи и полета.

Наследяването не е просто копиране на интерфейса на базовия клас:

- Обектите на производния клас съдържат в себе си подобект на базовия клас.
- Този подобект – все едно, че е създаден обект на самия базов клас.
- Отстрани, подобектът ще изглежда като обвит от обекта на наследения клас .

Явно извикване на методи на базовия клас

Възможно ли е super.super

```
class Time {
  protected int hour, minute;
   . . .
  public void printTime () {…}
}
```

```
class PreciseTime extends Time {
  private int second;

  public void printTime () { …}

  public void printTimeShort () {
     super.printTime();
  }
}
```

В метод от наследения клас искаме да извикаме препокрит метод на базовия клас.

Явно извикване на методи на базовия клас

Възможно ли е super.super

не

```
class Time {
  protected int hour, minute;
   . . .
  public void printTime () {…}
}
```

```
class PreciseTime extends Time {
  private int second;

  public void printTime () { …}

  public void printTimeShort () {
     super.printTime();
  }
}
```

Без наследяване

Може ли без наследяване

да

```
class Time { /* както досега */ }
```

```
class PreciseTime {
    private int hour, minute;
    private int second;
    public PreciseTime(...) ...
    public addMinutes(...)
    public void addSeconds(...)
    public void printTime()
}
```

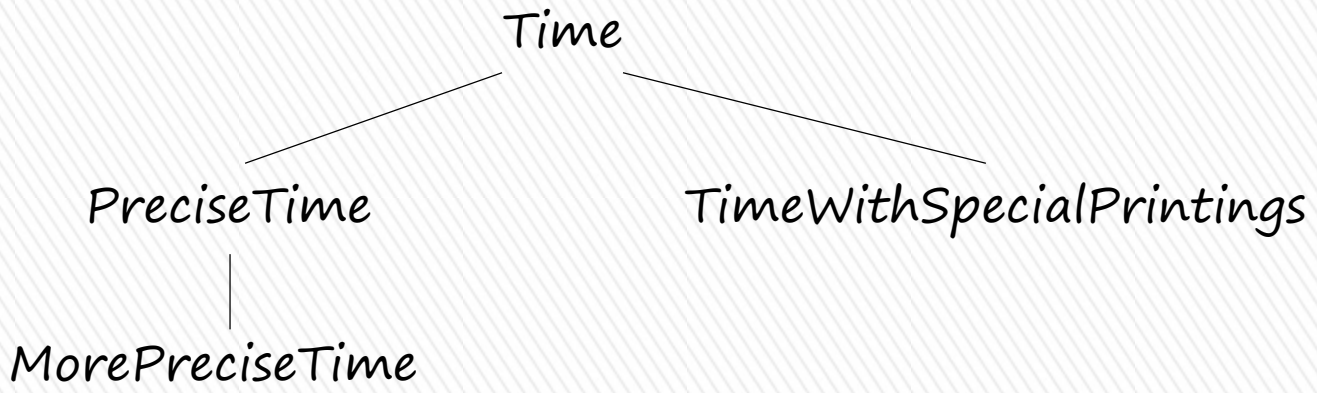
- Двете версии са еквивалентни! (можем без наследяване)
- По-дълги програми
  - Повтаряне на код:
    - \+ Промените са по-сложни, ако кодът, който ще се променя е двойно по-голям
  - Не се отразяват логическите отношения между класовете

Обобщение на контрола на достъп

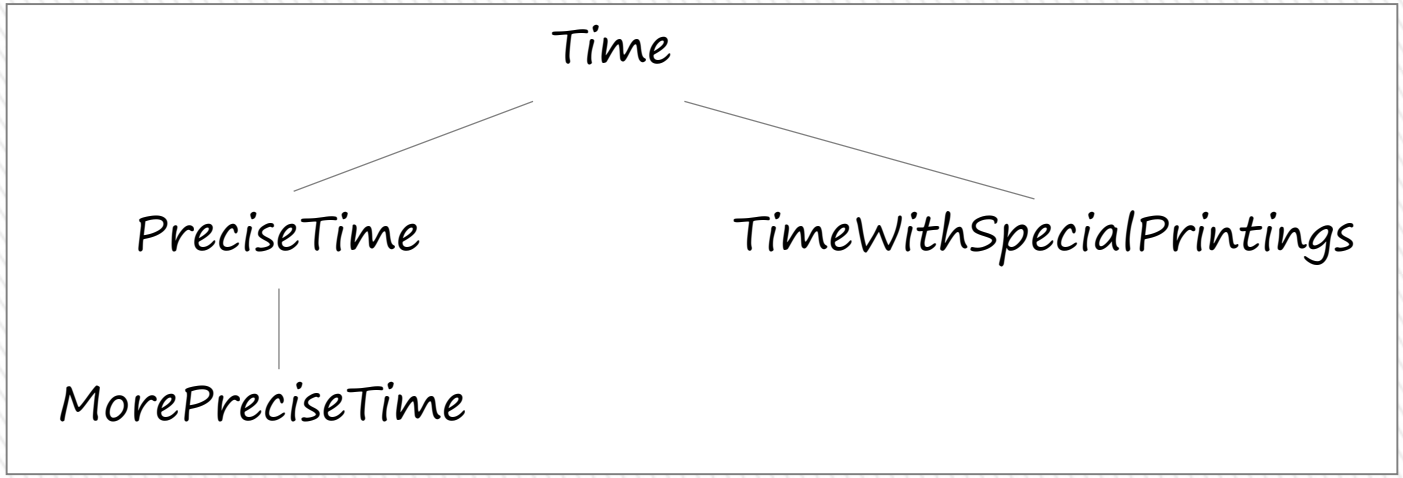
- Java доставя четири нива за контрол на достъпа до данни, методи и класове:
  - public достъп: Посредством методи от всички класове.
  - private достъп: Само посредством методи от собствения клас.
  - protected достъп: От собствения клас и неговите подкласове.
  - package достъп: От всички методи в собствения пакет - стандартна (по подразбиране) възможност.

Наследствени йерархии

```
class Time ...

class PreciseTime extends Time ...

class MorePreciseTime extens PreciseTime ...

class TimeWithSpecialPrintings extends Time ...
```



Наследствени йерархии



Суперкласове на MorePreciseTime:

PreciseTime (директен суперклас)

Time

Подкласове на Time:

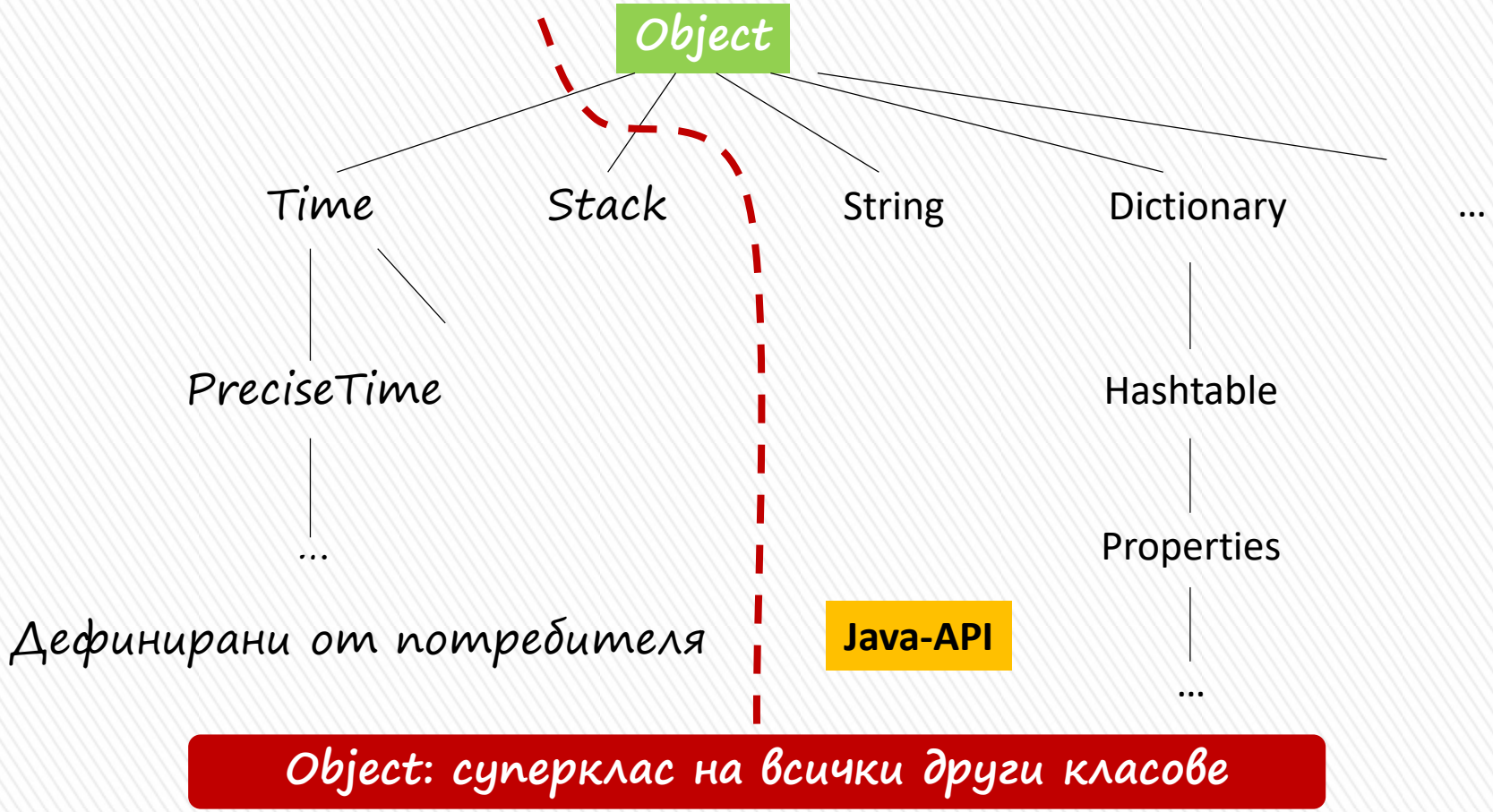
PreciseTime (директен подклас)

MorePreciseTime

TimeWithSpecialPrintings (директен подклас)

Клас Object

Object - суперклас на всички класове



API-Class java.lang.Object

java.lang

Class Object

`java.lang.Object`

public class **Object**

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

`Class`

Класът Object е коренът на всяка йерархията от класове.
Всеки клас има Object като суперклас.
За всички обекти са достъпни методите от този клас

Constructor Summary

`Object``()`	

Method Summary

protected `Object`	**`clone`**`()` Creates and returns a copy of this object.
boolean	**`equals`**`(Object obj)` Indicates whether some other object is "equal to" this one.
protected void	**`finalize`**`()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class`	**`getClass`**`()` Returns the runtime class of an object.
int	**`hashCode`**`()` Returns a hash code value for the object.
void	**`notify`**`()` Wakes up a single thread that is waiting on this object's monitor.
void	**`notifyAll`**`()` Wakes up all threads that are waiting on this object's monitor.
`String`	**`toString`**`()` Returns a string representation of the object.
void	**`wait`**`()` Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
void	**`wait`**`(long timeout)` Causes current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
void	**`wait`**`(long timeout, int nanos)` Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

```java
package equals;
public class ObjectClassDemo {
    Integer val;

    public ObjectClassDemo() {  }

    public ObjectClassDemo(Integer val) {
            this.val = val;     }

    public static void main(String[] args) {
            // създаване обекти от ObjectClassDemo
            ObjectClassDemo obj1 = new ObjectClassDemo(4577);
            ObjectClassDemo obj2 = new ObjectClassDemo(4577);
            System.out.println("Стринговият етикет на obj1 е "+obj1.val.toString());
            // проверка за еквивалентност на обекти
            if (obj1.equals(obj2)) {
                System.out.println("Обектите са еквивалентни");
            } else {
                System.out.println("Обектите не са еквивалентни");
            }
            obj2 = obj1; // присвояване референция obj1 на obj2
            // отново проверка за еквивалентност на обекти
            if (obj1.equals(obj2)) {
                System.out.println("Обектите са еквивалентни");
            } else {
                System.out.println("Обектите не са еквивалентни");
        }
    }
}
```

Properties ▪ Servers ▪ Data Source Explorer ▪ Snip

<terminated> ObjectClassDemo [Java Application] C

```
Стринговият етикет на obj1 е 4577
Обектите не са еквивалентни
Обектите са еквивалентни
```

Пример

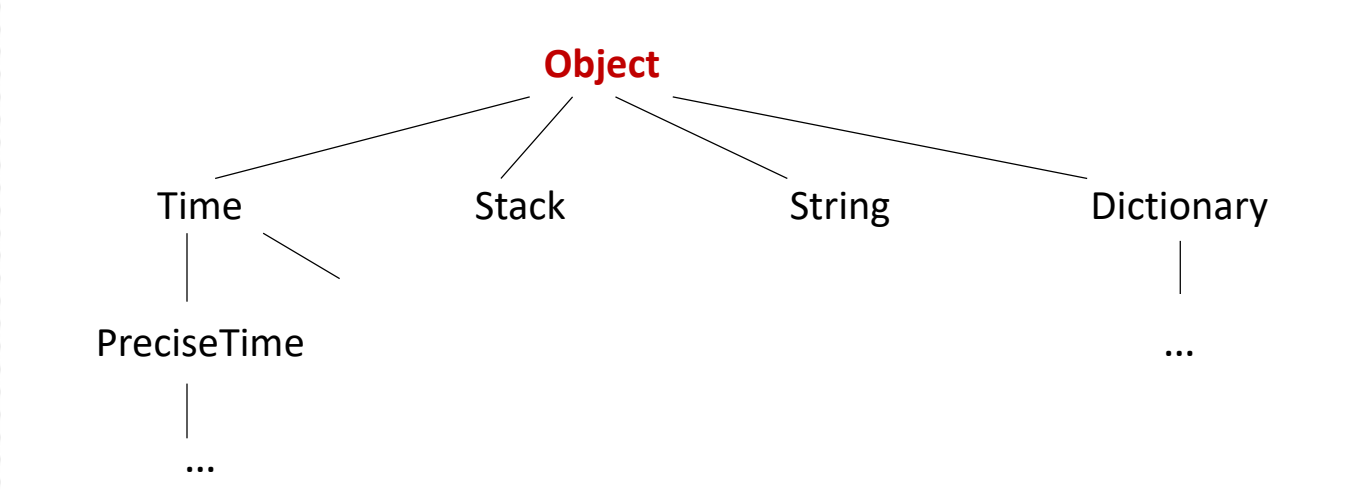
```
1  package equals;
2  import java.time.*;
4  public class Employee {
5      private String name; private double salary; private LocalDate hireDay;
6      public Employee(String name, double salary, int year, int month, int day) {
7          this.name = name;
8          this.salary = salary;
9          hireDay = LocalDate.of(year, month, day); }
10     public String getName() {
11         return name; }
12     public double getSalary() {
13         return salary; }
14     public LocalDate getHireDay() {
15         return hireDay; }
16     public void raiseSalary(double byPercent){
17         double raise = salary * byPercent / 100;
18         salary += raise; }
19     public boolean equals(Object otherObject) {
20         if (this == otherObject) return true;
21         if (otherObject == null) return false;
22         if (getClass() != otherObject.getClass()) return false;
23         Employee other = (Employee) otherObject;
24         return Objects.equals(name, other.name) && salary == other.salary
25                 && Objects.equals(hireDay, other.hireDay); }
26     public int hashCode() {
27         return Objects.hash(name, salary, hireDay); }
28     public String toString() {
29         return getClass().getName() + "[име=" + name + ",заплата=" + salary + ",плащане на=" + hireDay
30                 + "]"; }
31 }
```

```java
package equals;
public class Manager extends Employee {
    private double bonus;
    public Manager(String name, double salary, int year, int month, int day)
        super(name, salary, year, month, day);
        bonus = 0;
    }
    public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + bonus;
    }
    public void setBonus(double bonus) {
        this.bonus = bonus;
    }
    public boolean equals(Object otherObject) {
        if (!super.equals(otherObject)) return false;
        Manager other = (Manager) otherObject;
        // super.equals checked that this and other belong to the same class
        return bonus == other.bonus;
    }
    public int hashCode() {
        return super.hashCode() + 17 * new Double(bonus).hashCode();
    }
    public String toString() {
        return super.toString() + "[бонус=" + bonus + "]";
    }
}
```

```
Properties  Servers  Data Source Explorer  Snippets  Problems  Console
<terminated> EqualsTest [Java Application] C:\Program Files\Java\jre1.8.0_341\bin\javaw.exe  (16.12.2025 11:44:02 – 11:44:02)
alice1 == alice2: true
alice1 == alice3: false
alice1.equals(alice3): true
alice1.equals(bob): false
alice1.toString(): equals.Employee[име=Алиса Прекрасна,заплата=2500.0,плащане на=2025-12-15]
bob.toString(): equals.Employee[име=Баш Майстор,заплата=3500.0,плащане на=2025-12-10]
boss.toString(): equals.Manager[име=Крали Карлов,заплата=8000.0,плащане на=2025-12-01][бонус=3000.0]
carl.equals(boss): false
alice1.hashCode(): 1160346888
alice3.hashCode(): 1160346888
bob.hashCode(): 345782330
carl.hashCode(): 966168830
```

```java
package equals;

public class EqualsTest {
        public static void main(String[] arg
        Employee alice1 = new Employee("Алиса Прекрасна", 2500, 2025, 12, 15);
        Employee alice2 = alice1;
        Employee alice3 = new Employee("Алиса Прекрасна", 2500, 2025, 12, 15);
        Employee bob = new Employee("Баш Майстор", 3500, 2025, 12, 10);

        System.out.println("alice1 == alice2: " + (alice1 == alice2));
        System.out.println("alice1 == alice3: " + (alice1 == alice3));
        System.out.println("alice1.equals(alice3): " + alice1.equals(alice3));
        System.out.println("alice1.equals(bob): " + alice1.equals(bob));

        System.out.println("alice1.toString(): " + alice1);
        System.out.println("bob.toString(): " + bob);

        Manager carl = new Manager("Крали Карлов", 8000, 2025, 12, 1);
        Manager boss = new Manager("Крали Карлов", 8000, 2025, 12, 1);
        boss.setBonus(3000);

        System.out.println("boss.toString(): " + boss);
        System.out.println("carl.equals(boss): " + carl.equals(boss));
        System.out.println("alice1.hashCode(): " + alice1.hashCode());
        System.out.println("alice3.hashCode(): " + alice3.hashCode());
        System.out.println("bob.hashCode(): " + bob.hashCode());
        System.out.println("carl.hashCode(): " + carl.hashCode());
    }
}
```

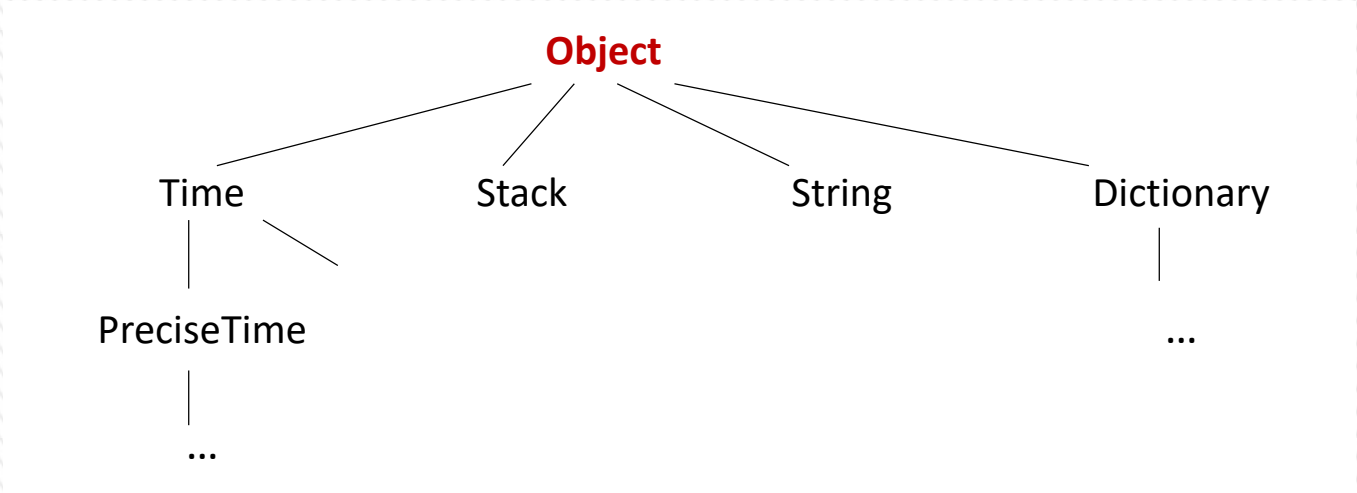
Правила за съвместимост



Обектите на един подклас могат да стоят там, където са допустими обекти на супекласовете.

```
Time t;
t = new PreciseTime(12, 10, 1);
t.printTime();
```

Съвместимост с Object



Извод: променливите на тип Object могат да притежават като стойност инстанции на произволни класове

```
Object o1, o2;
o1 = new PreciseTime(12, 10, 1);
o2 = new Stack(100);
```

Идея за полиморфизъм

Полиморфизъм: основна идея

Полиморфизъм: “много форми на появяване“

```
 Time t1;
PreciseTime t2;

t1 = new Time(12, 10);
t2 = new PreciseTime(0, 10, 1);
t1.printTime();
t2.printTime();
```

Полиморфизъм:

- Подобни операции с една и съща сигнатура (име, брой и тип на параметри)
- Поведението може да варира в зависимост от актуалния тип на обекта

Повторение: Overloading = едно и също име за методи с различни списъци на параметрите

Полиморфизъм: избор на правилния метод

Кой избира – компилатор или JVM?

```
Time t1;
PreciseTime t2;

t1 = new Time(12, 10);
t2 = new PreciseTime(0, 10, 1);
t1.printTime();
t2.printTime();
```

Динамично свързване

От Time или PreciseTime?

```
Time t;
if (readValue() == 'T')
   t = new Time(12, 10);
else
   t = new PreciseTime(12, 10, 1);
t.printTime();
```

Компилаторът не може да реши!

Избор на правилния метод едва в run-time (по време на изпълнение на програмата) възможно.

Полиморфизъм: избор на правилния метод

Кой избира – компилатор или JVM?

JVM

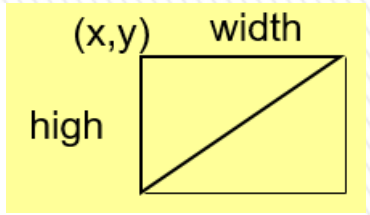
```
Time t1;
PreciseTime t2;

t1 = new Time(12, 10);
t2 = new PreciseTime(0, 10, 1);
t1.printTime();
t2.printTime();
```

Изборът на оператор, зависим от целевия обект (напр. t1, t2).

Абстрактни класове

Абстрактни класове

```
abstract class Figure {
    protected int x, y, width, high;
    public void setx(int xNew) {
        x = xNew;
    }
    // setY, setWidth, setHigh
    public abstract float calculateSquare();
}
```



Абстрактните класове съдържат методи без реализация.

calculateSquare метод без реализация:

- Реализацията трябва да се извърши в подкласа:

  Figure → правоъгълник, триъгълник, …

- Не е разрешено създаване на инстанции на класа.
- Смисъл: логическо структуриране на множества от класове.

Мотивация за абстрактни класове

- Когато разширяваме съществуващ клас имаме избора дали да предефинираме методи на суперкласа.
- В определени случаи е желателно да заставим програмистите да предефинират определени методи.
  - > Напр. не съществува добър default (по подразбиране) за суперкласа
  - > Само програмистът на подкласа знае каква е подходящата реализация на метода.

Пример

```
1  package abstractClasses;
2
3  public abstract class Person {
4      public abstract String getDescription();
5      private String name;
6
7      public Person(String name){
8          this.name = name;
9      }
10
11     public String getName() {
12         return name;
13     }
14 }
```

Пример

```
1  package abstractClasses;
2  import java.time.*;
3
4  public class Employee extends Person {
5      private double salary;
6      private LocalDate hireDay;
7      public Employee(String name, double salary, int year, int month, int day) {
8          super(name);
9          this.salary = salary;
10         hireDay = LocalDate.of(year, month, day);
11     }
12     public double getSalary() {
13         return salary;
14     }
15     public LocalDate getHireDay() {
16         return hireDay;
17     }
18     public String getDescription() {
19         return String.format("служител със заплата от %.2f", salary);
20     }
21     public void raiseSalary(double byPercent) {
22         double raise = salary * byPercent / 100;
23         salary += raise;
24     }
25 }
```

Пример

```
1  package abstractClasses;
2
3  public class Student extends Person {
4      private String major;
5
6      public Student(String name, String major) {
7          // pass n to superclass constructor
8          super(name);
9          this.major = major;
10     }
11
12     public String getDescription() {
13         return "студент по специалност " + major;
14     }
15 }
```

Пример

```
1  package abstractClasses;
2
3  public class PersonTest {
4      public static void main(String[] args) {
5          Person[] people = new Person[2];
6
7          // инициализира масива people с Student и Employee обекти
8          people[0] = new Employee("Иван Иванов", 5000, 2025, 11, 30);
9          people[1] = new Student("Божур Цветков", "'софтуерно инеженерство'");
10
11         // извежда names и descriptions на всички Person обекти
12         for (Person p : people)
13             System.out.println(p.getName() + ", " + p.getDescription());
14     }
15 }
```

Properties ▪ Servers ▪ Data Source Explorer ▪ Snippets Problems Console

<terminated> PersonTest [Java Application] C:\Program Files\Java\jre1.8.0_341\bin\javaw.exe (16.1

```
Иван Иванов, служител със заплата от 5000,00
Божур Цветков, студент по специалност 'софтуерно инеженерство'
```

Видове класове

- Абстрактен клас:
  - > За който не можем да създаваме обекти.
  - > Ключова дума abstract.
  - > Клас, който дефинира един абстрактен метод или който наследява абстрактен метод без да го препокрива.
- Конкретен клас:
  - > Можем да създаваме обекти.
  - > Може да има обектни референции, типът на които са абстрактни класове.

Final: ограничение на полиморфизма

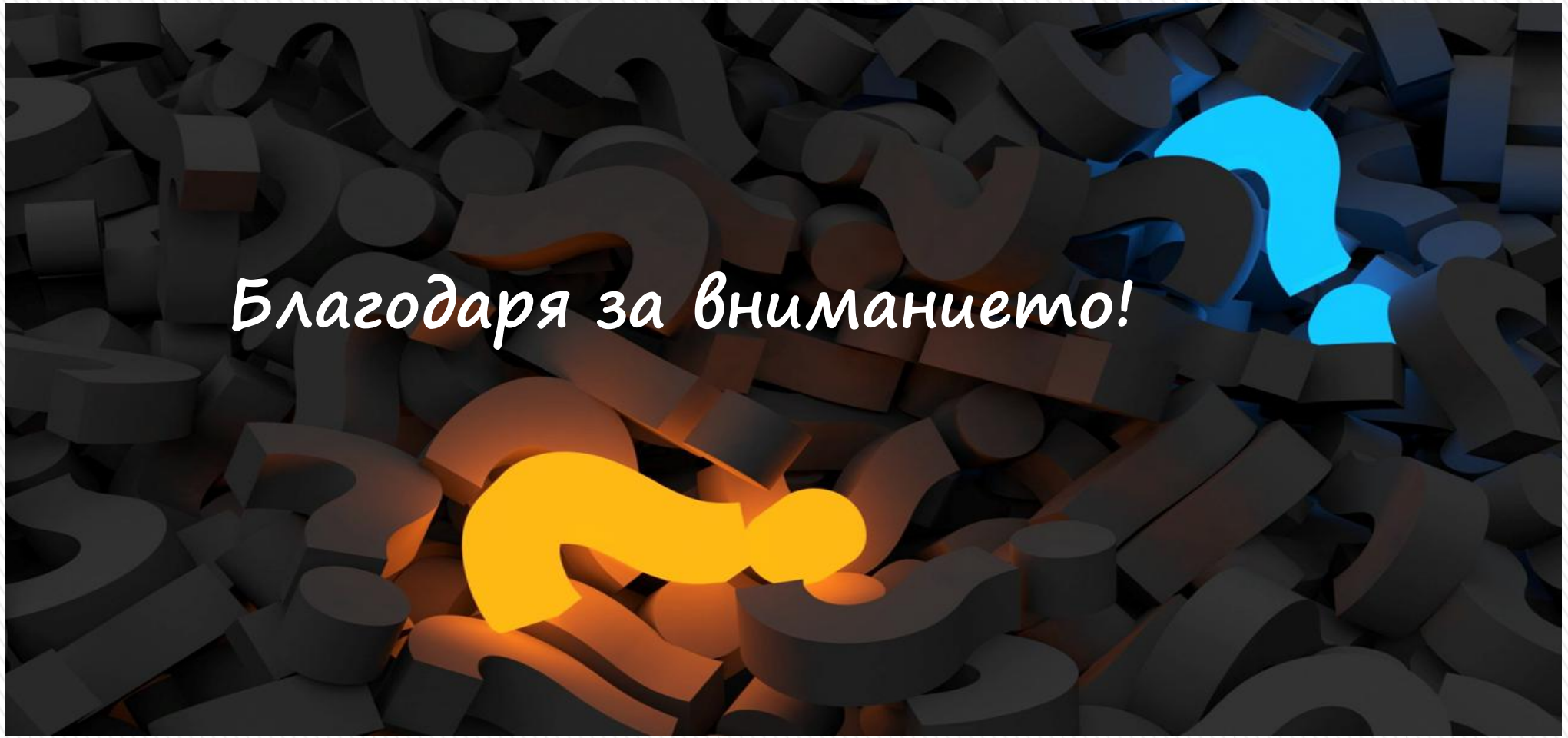
```
class Time {
    public final void addMinutes(...) {
        ...
    }
    public void printTime()
}
```

```
public final class String;
```

- Методите не могат да бъдат предефинирани в подкласа.
- Идея: грижа за стабилна семантика (сигурност).
- Класовете не могат да бъдат разширявани:
  напр. Java-API.

Мотивация за final

- Обратна на абстрактните класове логика:
  - Избягваме други програмисти да създават подкласове или да препокриват методи.
  - Примери:
    - public final class String { … }
    - public final boolean checkPassword(String password)


Благодаря за вниманието!